NASA Technical Memorandum 104380

RSM 1.0 User's Guide

A Resupply Scheduler Using Integer Optimization

Larry A. Viterna and Robert D. Green
Lewis Research Center
Cleveland, Ohio

and

David M. Reed
Wittenberg University
Springfield, Ohio

May 1991

NASA

RSM 1.0 User's Guide
A Resupply Scheduler Using Integer Optimization

Larry A. Viterna and Robert D. Green
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

and

David M. Reed*
Wittenberg University
Springfield, Ohio 45501

Abstract

RSM (Resupply Scheduling Model) is a PC based, fully menu–driven computer program. It uses integer programming techniques to determine an optimum schedule to replace components on or before a fixed replacement period, subject to user defined constraints such as transportation mass and volume limits or available repair crew time.

Principal input for RSM includes component properties such as mass and volume and an assembly sequence. Resource constraints are entered for each period corresponding to the component properties.

Though written to analyze the electrical power system on the Space Station Freedom, RSM is quite general and can be used to model the resupply of almost any system subject to user defined resource constraints.

This report presents a step by step procedure to preparing the input, performing the analysis and interpreting the results. Instructions for installing the program and information on the algorithms are contained in the appendices.

*Summer Student Intern at NASA Lewis Research Center.

Contents

Introduction

NASA has extensive experience in the development of highly reliable spacecraft. Only a very few of these spacecraft have been maintained by scheduled resupply flights from Earth. Other government agencies, particularly the military, have years of experience in logistics analysis and management. Computer programs developed for those applications, however, are not readily adaptable to the Space Station Freedom (SSF). For example, scheduling algorithms for support of a fleet of vessels often use standard linear programming algorithms which can be inaccurate for small quantities of components needed to support a single vessel.

RSM (Resupply Scheduling Model) is a PC based, fully menu–driven computer program. It uses integer programming (IP) techniques to determine an optimum schedule to replace components on or before a fixed replacement period, subject to constraints such as transportation mass and volume limits on the Space Transportation System (STS) or available astronaut extravehicular activity (EVA) time.

Principal input for RSM includes component properties such as mass and volume and an assembly sequence. Constraints are entered for each period corresponding to the component properties.

Though written to analyze the electrical power system (EPS) on the Space Station Freedom, RSM is quite general and can be used to model the resupply of almost any system subject to user defined resource constraints.

Hardware and software requirements and installation instructions for RSM are given in Appendix A.

Nomenclature

APL	A Programming Language
EPS	Electrical Power System on SSF
EVA	Extravehicular Activity
IP	Integer Programming (Optimization) Problem
RSM	Resupply Scheduling Model (computer program)
SSF	Space Station Freedom
STS	Space Transportation System

Preparing and Managing the Input

The user interface to RSM consists of a cursor/keystroke driven menu along the top of the screen, a data input editor and a file management system. After starting RSM the title screen appears followed by the menu screen shown in Figure 1.

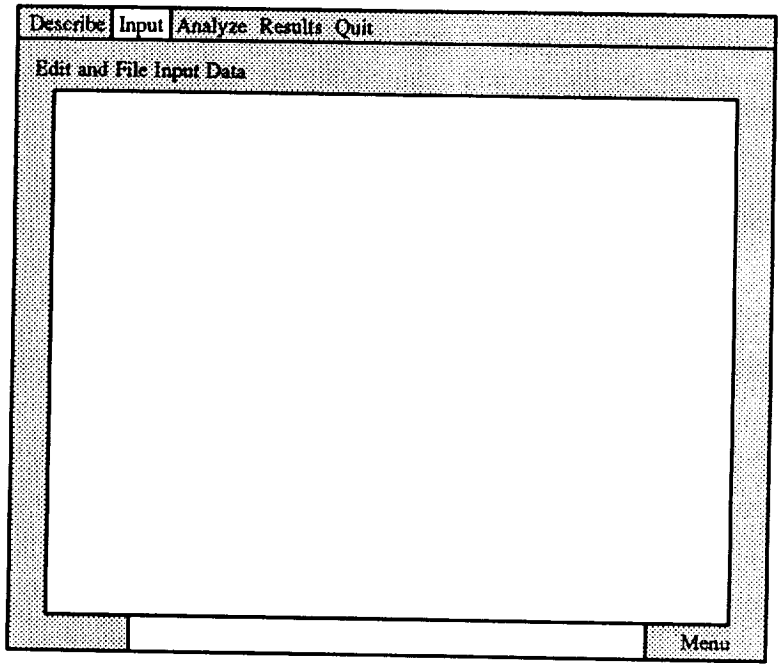


Figure 1: RSM Main Menu Screen

Menu choices are selected by using left and right cursor motion followed by the [Enter] key. Alternatively, the first character of the selection can be typed with immediate execution of that selection.

Resources

Choosing the Input option displays the menu shown in Figure 2. Pressing the [Enter] key will select the input option and display the menu screen shown in Figure 3. Pressing the [Enter] key again will select the edit Resources option and display the data input screen shown in Figure 4. In the first field enter the length (or duration) of one period. This can be a real number in units such as years or some other time unit as long as it is the same as is used to define component lifetimes. In the remaining input fields enter unique names for each of the available resources. In the example shown, mass and labor are defined as the resources of interest.

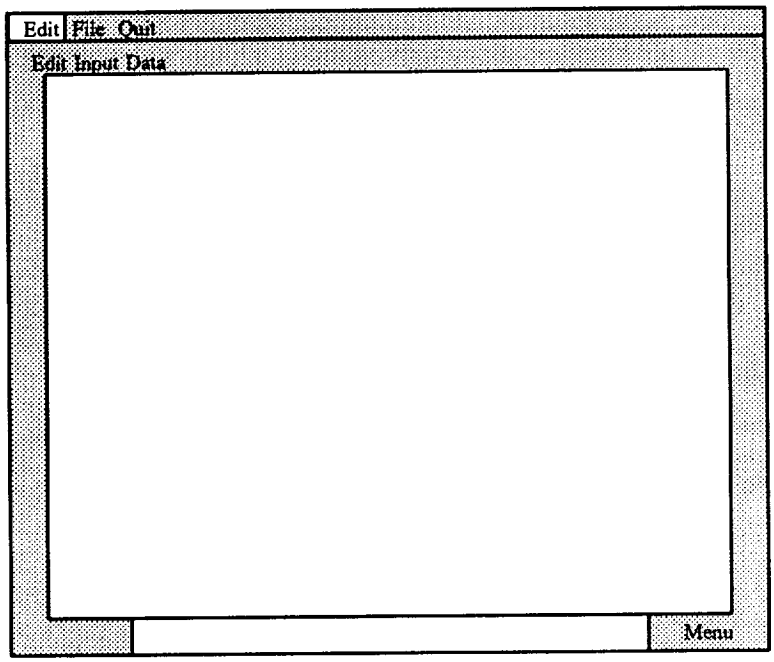


Figure 2: Input Menu Screen

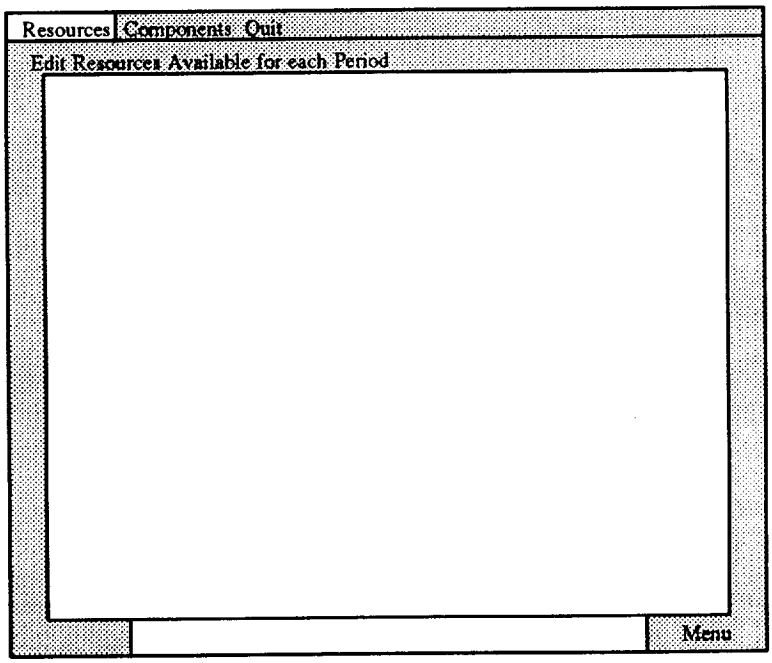


Figure 3: Resources and Components Input Edit Menu Screen

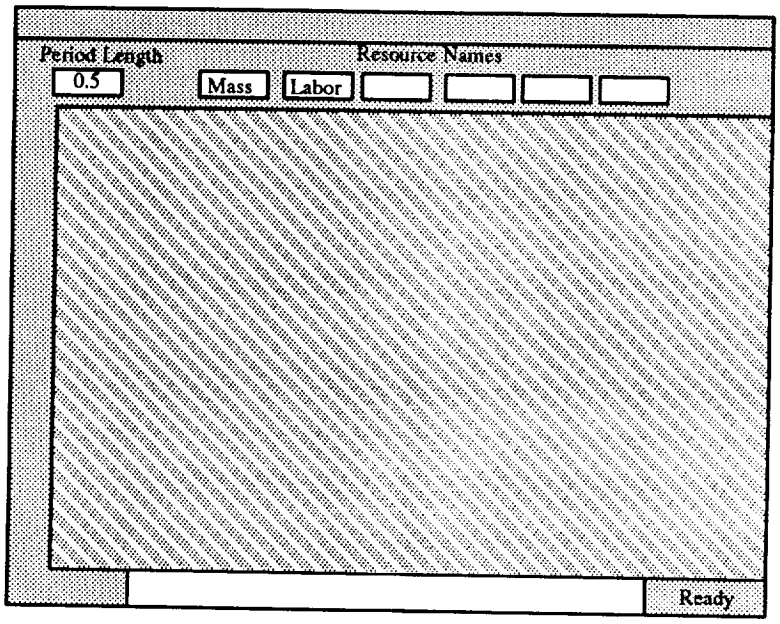


Figure 4: Period Length and Resource Name Input Screen

Press the [Enter] key when finished and the input screen shown in Figure 5 will appear. Enter the available resources for each period. In the example shown, the mass limit of the resupply transportation system is 100 for the first three periods and 70 for the remaining periods. Note that the last row which contains data determines the number of periods of the scheduling analysis. In the example shown there are 16 periods of length 0.5 for a total duration of 8.0.

It is important that resource allocations be sufficient to allow the assembly sequence installations to be met.

Pop up Help screens for the input editors of Figures 4 and 5 are available by pressing function key [F1]. The [Tab] key positions the cursor to the next field. The [Esc] key exits the input editor without saving the data. Further details of the use of the editor are contained in Appendix B.

Component Properties

Selecting the Components input option from Figure 3 displays the Component menu screen of Figure 6. Selecting the Properties option displays the input screen of Figure 7.

Each column is for a different resource. In addition to the lifetime constraints that are always created, the input in these columns correspond to additional resource constraints. An

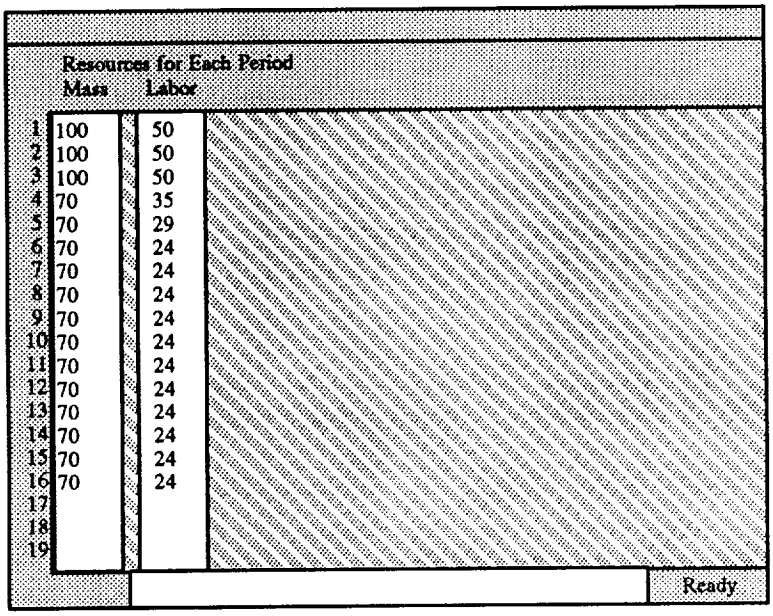


Figure 5: Resource for Each Period Input Screen

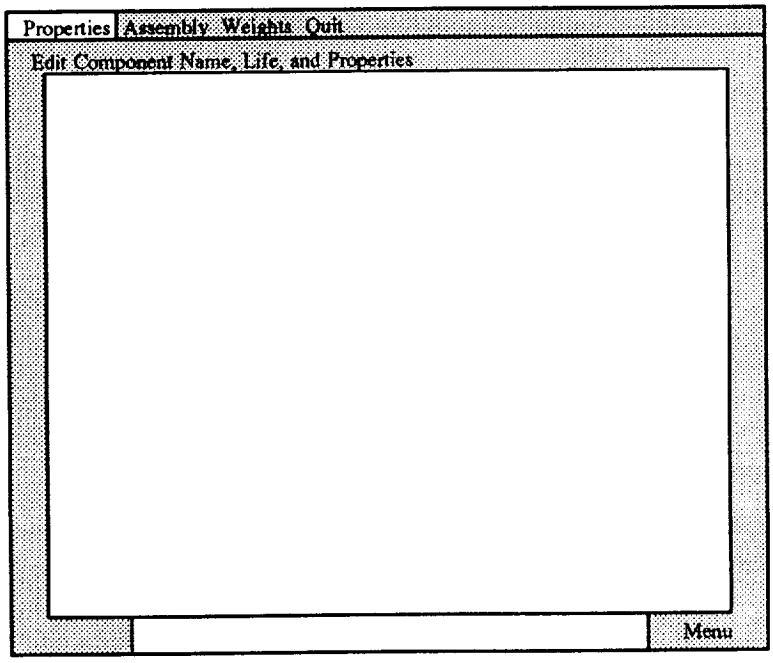


Figure 6: Component Input Menu Screen

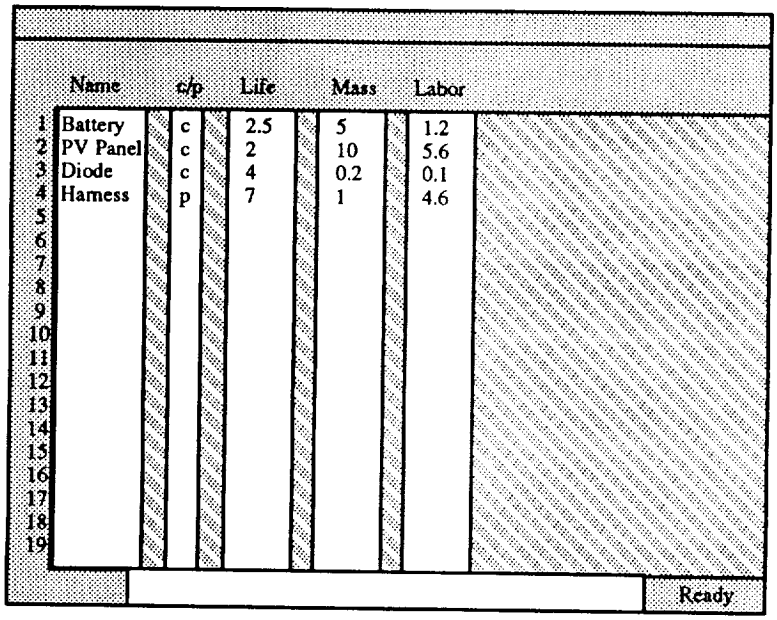


Figure 7: Component Property Input Screen

example of this is a mass constraint. Assume there is a limit to the total amount of mass of replacement parts that can be delivered each period; each component has a mass and therefore constraints are created to insure that the sum of the number of replaced units of each component multiplied by its mass is less than or equal to the allotted amount in the resource input screen earlier.

In column 1 of the properties input editor enter a unique component name.

In column 2 enter "c" or "p" to designate whether this component is to be calculated or prescribed. A calculated component's schedule is determined from the integer program analysis. For a prescribed component, the schedule is fixed such that the component is replaced exactly when its lifetime expires.

In column 3 enter the component's lifetime or critical replacement time. It has units such as years which correspond to the period duration under the resource input editor.

In the remaining columns enter the amounts used by one of these components for each of the resources.

RSM can be used to model many types of systems depending on how the components and resource constraints are set–up. The reason for prescribing some components is to decrease the size of the internal resource constraint matrix. This shortens the time required to solve the problem and requires less computer memory.

Some systems have many components that should be prescribed while other systems will not have any. The following are some general rules for deciding which components to prescribed:

Components which there are relatively few.

Components which have constraint coefficients which are an order of magnitude smaller than other components.

Components which have a much longer lifetime than the other components.

Components that meet at least two of these rules or one rule and are not too far from meeting another rule probably should be prescribed. For these components, a fixed schedule will be assumed. The program automatically subtracts the required allocation for each additional constraint in the periods in which they are replaced.

Assembly Sequence

Selecting the Assembly option of Figure 6 displays the assembly sequence input screen of Figure 8. The number of components installed during each period are entered here. For

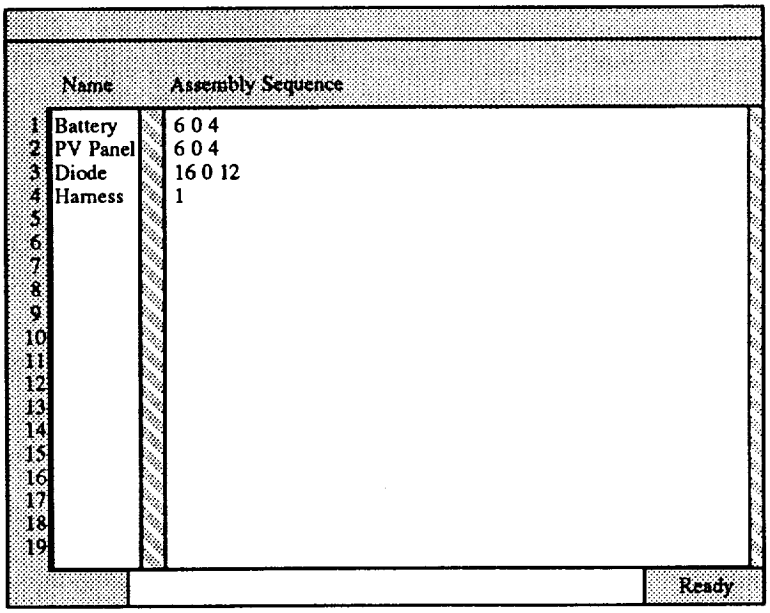


Figure 8: Component Assembly Sequence Input Screen

example as shown in the figure, six batteries are installed in the first period, none in the second, 4 in the third, and none later.

RSM's internal algorithms have the following restrictions:

1. During the component's assembly sequence, no replacements can take place. Thus, each component's lifetime must be longer than its assembly sequence length.

Note: If a replacement is to take place during the assembly sequence, ignore the initial installation of the component and make what would be the replacement during the assembly sequence, the initial installation (i.e. if a component has an assembly sequence of 2, 0, 0 and the component needs to be replaced in period 3, input the assembly sequence as 0, 0, 2.

2. The number of components brought up during the assembly sequence determines how many units must be operational at all times.

3. Each unit must be replaced before or during the period which is the period in which the component was installed plus the lifetime of the component.

Weighting Factor

Selecting the Weighting Factor option of Figure 6 displays the assembly sequence input screen of Figure 9. This input defines the coefficients in the objective function of the internal integer program solver. Nominally these are all set to a value of 1 for equal weighting of all components. This is used to minimize the total number of components replaced.

It is possible to use the objective coefficients to investigate a number of interesting problems. For example by assigning every component a coefficient corresponding to its cost, the total cost can be minimized.

Input File Management

Selecting the File option from the input menu screen shown in Figure 2 displays the file management menu screen of Figure 10. Selecting the Save option displays the file save screen of Figure 11.

RSM supports long filenames. The list of previously stored input files are displayed in reverse chronological order according to the times there were saved. The size, date, and time saved are displayed along with the names. Near the top of the screen, RSM prompts for

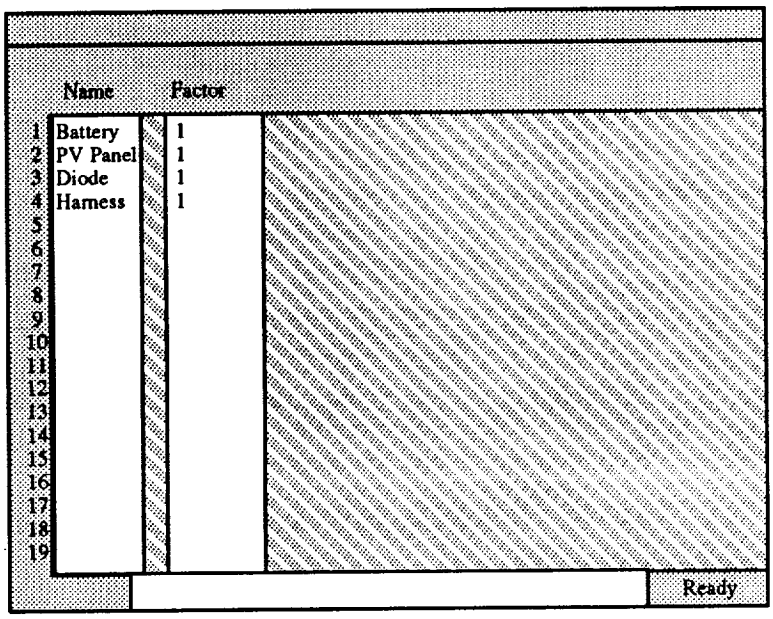


Figure 9: Component Weighting Factors Input Screen

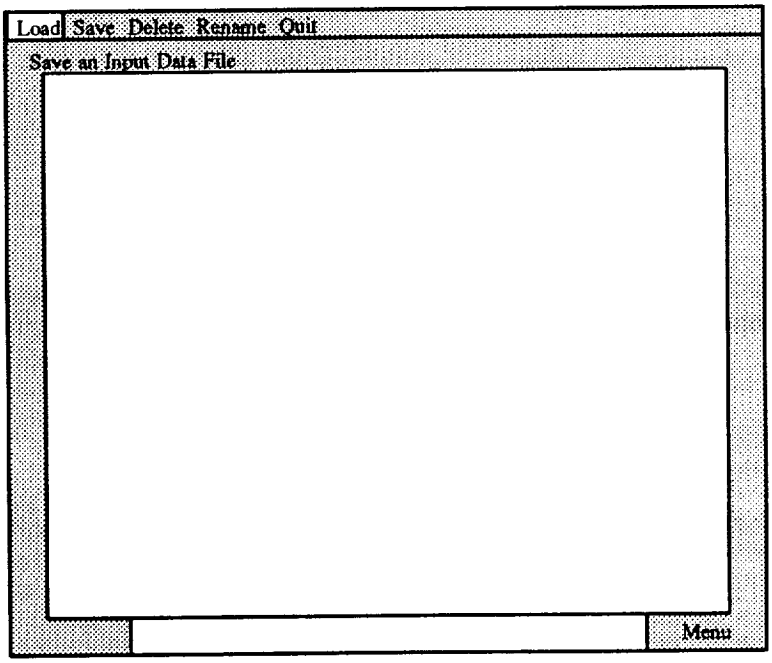


Figure 10: Input File Management Input Menu Screen

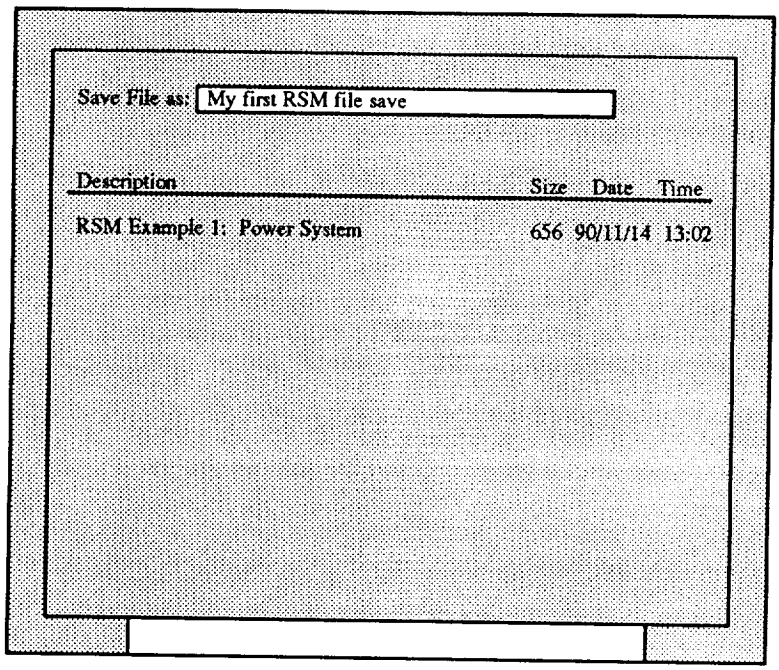


Figure 11: File Save Screen

a name to store the current input. The name can consist of any alphanumeric characters, with the exception that the first character must be alphabetic. Pressing the [Enter] key stores the current input data. Pressing the [Esc] key exits all file menus without completing the file operation.

Selecting the Load option displays the file load screen of Figure 12. Again RSM displays the list of previously stored input data. The specific file is selected with the up and down cursor keys and pressing [Enter] when the desired file is highlighted.

A file can be renamed similarly by first selecting the old file name using the cursor keys and then entering the new name at the prompt.

A file can also be deleted by selecting and pressing [Enter]. To prevent deleting a file, the [Esc] key must be pressed before leaving the delete file menu.

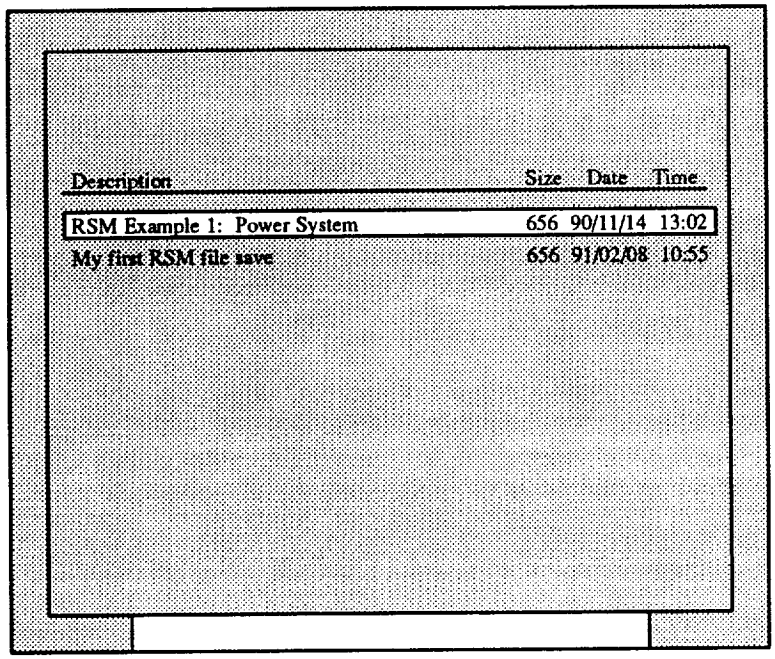


Figure 12: File Load Screen

Performing the Analysis

Once the input has been entered or loaded from a previously stored system, the analysis can be performed. Choosing the Analyze option in the menu in Figure 1 begins the scheduling analysis. The status of the iterative solution is displayed during the analysis. Information includes the number of iterations, the time per iteration, and the total time since the analysis began. In addition, a message gives the first period in which resources are being exceeded on each iteration. If the scheduling problem is unfeasible the final message will also indicate the first period in which the resources were exceeded.

For the example discussed earlier the final status screen is shown in Figure 13. The times shown are for a Dell System 425E personal computer.

Press the [Enter] key to return to the main menu.

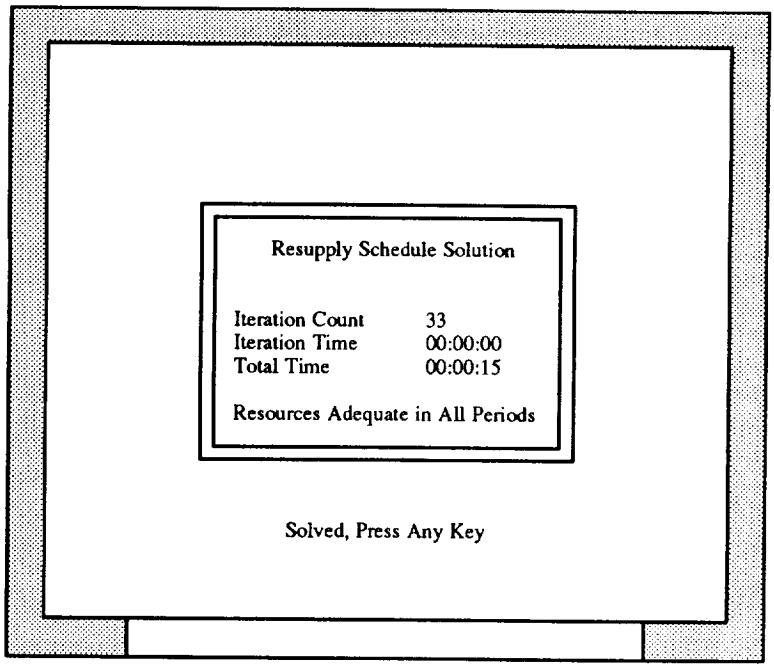


Figure 13: Solution Iteration Final Status Screen

Interpreting and Managing the Results

Once a scheduling problem has be solved, the results can be displayed, printed, or filed. Choosing the Results option from the menu shown in Figure 1 will display the menu screen shown in Figure 14.

Displaying Resource Usage

Selecting the Display option will display the menu screen shown in Figure 15. Selecting the Resource option will display the resource usage screen shown in Figure 16. The quantity of the first resource (mass) is shown for each period as a bar chart. Also shown, as a dashed line, are the resource constraint levels specified in the input. Other resource usage can be viewed by using the [Page Down] keys.

Note that these display screens use text based bar charts at this time and thus are somewhat deficient in resolution. For exact values, use the print option to produce numeric results.

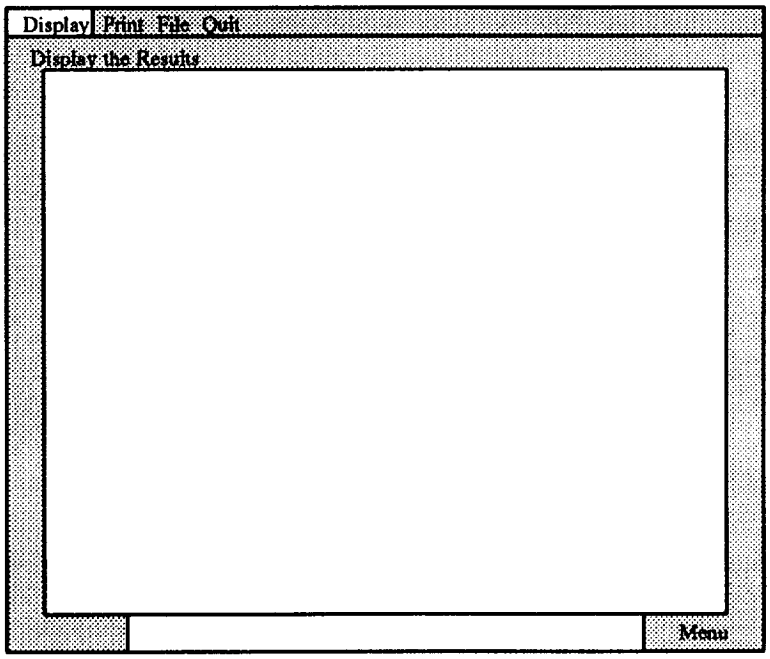


Figure 14: Results Management Menu Screen

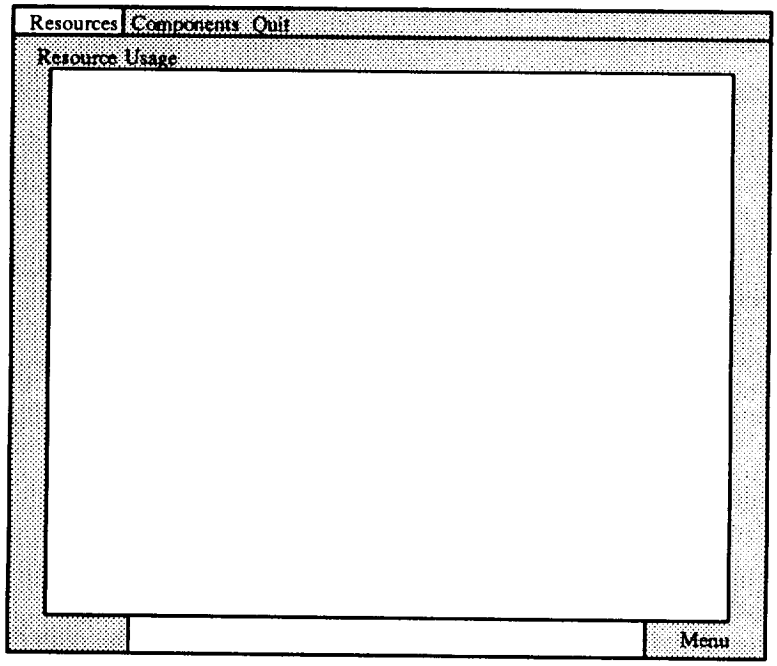


Figure 15: Resources and Components Results Display Menu Screen

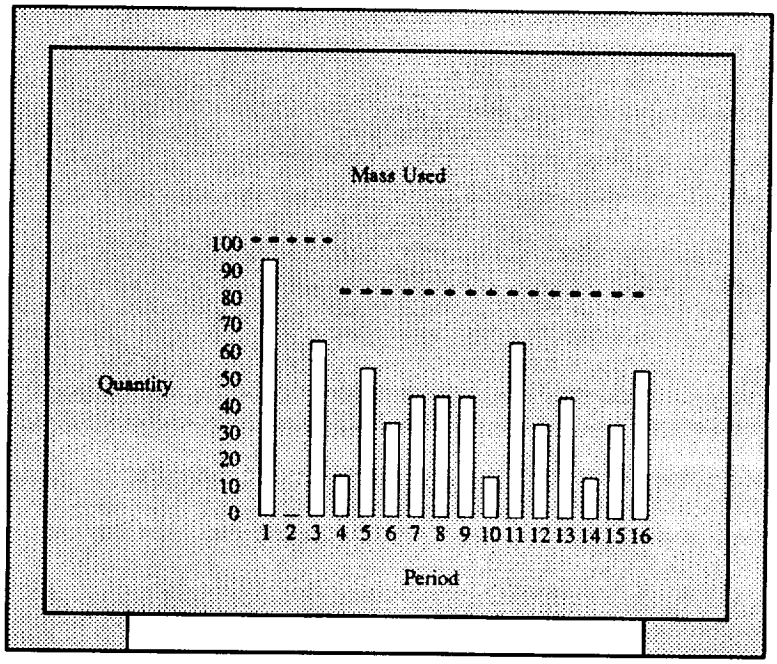


Figure 16: Display of Resource Usage

Displaying Component Resupply Schedules

Selecting the Components option will display the component resupply schedules screen shown in Figure 17. The quantity of the first component (Battery) is shown for each period as a bar chart. Other component resupply schedules can be viewed by using the [Page Down] keys.

Printing the Results

Selecting the Print option of Figure 14 will print the usage of all resources and the resupply schedules for all components. The numerical results for the first 7 periods are shown in Figure 18.

Results File Management

Selecting the File option of Figure 14 displays the file management menu screen of Figure 10. Loading, Saving, Deleting, and Renaming options are identical to the Input File Management system discussed earlier.

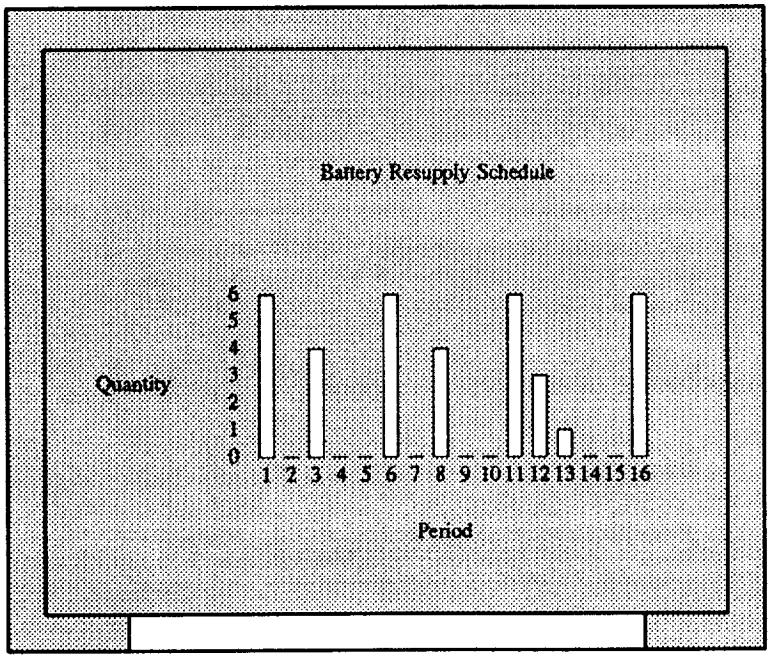


Figure 17: Display of the Component Resupply Schedule

Resource Usage

Period	1	2	3	4	5	6	7
Mass	94.200	.000	62.400	10.000	50.000	30.000	40.000
Labor	47.000	.000	28.400	5.600	28.000	7.200	22.400

Component Resupply

Period	1	2	3	4	5	6	7
Battery	6	0	4	0	0	6	0
PV Panel	6	0	4	1	5	0	4
Diode	16	0	12	0	0	0	0

Figure 18: Numerical Results for the First Seven Periods

Appendix A: Computer Requirements and Installation

RSM is provided on 5.25 inch. high density floppy diskettes containing the following five files:

rsm.exe

rsm.atf

example_.rsf

example_.rrf

rsf.dir

The two "example" files can be read from within RSM and contain the system configuration and analytical results from the example given in this users guide.

There are two possible ways of running RSM, depending on which version of the program is being used.

RSM can exist as a stand–alone program from DOS. In this "packed" version of RSM, only the memory within the 640K limit of DOS is available.

If RSM is contained as an IBM APL2/PC workspace (as from the original development), the workspace must be loaded from transfer format. The advantage to using RSM from within the APL2 environment is that the program can make full use of any installed extended memory.

Stand–alone EXE Version

For this version, RSM requires an IBM PC or compatible with at least 640 K of memory and DOS 3.1 or higher.

To install RSM, simply create a directory and copy the rsm.exe file to it.

To run RSM, change to the directory containing the rsm.exe file and type "RSM" at the DOS prompt.

APL2/PC Workspace Version

For this version, RSM requires the IBM APL2 programming language on a 80386 or 80486 based microcomputer with an 80387 math coprocessor, at least 2 megabytes of extended memory, and DOS 3.3 or higher. APL2 for the PC is available from:

IBM Direct
Phone: 800–IBM–2468
Part Number 6242936

To install, copy the RSM workspace, "rsm.atf", to the system APL2 directory. Next, enter the APL2 system. It should be noted that RSM requires seven auxiliary processors: ap2, ap80, ap100, ap101, ap103, ap124, and ap210. An example invocation would be:

```
apl232 ap2 ap80 ap100 ap101 ap103 ap124 ap210
```

RSM can now be imported from its transfer–format file with the command:

```
)IN RSM
```

From here, the user may examine or modify any of the RSM code. To begin execution of RSM, the main function is called by typing:

```
RSM
```

RSM can also be stored in workspace form with the command:

```
)SAVE RSM
```

All further use of RSM can now use the command:

```
)LOAD RSM
```

Appendix B: The Data Input Full Screen Editor

This appendix contains instructions for using the data input full screen editors. These instructions may be viewed during the editor session by pressing the Help key, [F1]. The help window disappears upon pressing [Escape]. The "Function Key" numbering and corresponding keystrokes are as follows:

Function Keys

Key Number	Keystrokes
F1 – F10	[F1] – [F10]
F11 – F20	[Shift]/[F1] – [F10]

Quit Editing Session

To quit the editing session without saving the changes you made, press [Esc].

Save Editing Session

To exit the editing session and save these changes, press [Enter].

Cursor Movement

Use the [Tab] key to move the cursor to the right from one column to the next. Use [Shift]/[Tab] to move the cursor left to the previous column. Press the "Large Plus", [+], key to jump to the next line of the table. Press [F5] to jump to the top row of the table. Press [F6] to jump to the bottom row of the table.

Restore Original Data to Screen

To restore the original contents at the cursor position, press [F3]. Press [Shift]/[F3] to restore the entire row.

Insert a Row

To insert a row on the screen, place the cursor one line below where the new line is to be located and press [F8]. At the prompt, type in the number of rows to be inserted and press the [Enter] key.

Toggle between Replace and Insert Modes

At the beginning of an edit session, you will be in Replace mode, in which keystrokes will write over the previous text. The editor can also operate in Insert mode, in which the keystrokes will move existing text to the right. To switch from one mode to the other, press the [Insert] key.

Mark/Unmark Rows

To **mark** rows of the table for copying, moving, deleting or saving, move the cursor to each of the desired rows and press [Shift]/[F9]. Each of the rows will be highlighted. Marked (highlighted) rows may be copied, moved or deleted. To **unmark** a row, move the cursor to that row and press [Shift]/[F9]. To mark or unmark **all** rows, press [Shift]/[F10].

Delete Rows

To delete a row, place the cursor on the row to be deleted and press [Shift]/[F8]. If any rows are marked (highlighted), these rows will be deleted.

Copy Rows

To copy a row of the table, move the cursor to the row and press [F10]. If any rows are marked (highlighted), these rows will be copied.

Save Rows

To save rows of the table for later use, mark the rows (see the paragraph above entitled **Mark/Unmark Rows**), press [Shift]/[F6] and enter a name for this group of rows.

Retrieved Saved Rows

Rows saved as described above can be retrieved into an editing session by pressing [Shift]/[F5] at the cursor and entering a name.

Discard Saved Rows

To discard a set of rows, press [Ctrl]/[F6].

Print

To use the printer while in an editor session, press [F2]. The following options will be available with a single keystroke:

[G] 'Go': Prints any marked rows. If no rows are marked, the entire table will be printed.

[R] Resets line counter to 0. This aligns the printer to the top of the page.

[L] Advance the printer 1 line.

[P] Advance the printer 1 page.

Appendix C: Integer Optimization Algorithm

This appendix is intended for those who wish to have a better understanding of how the problem set–up translates into a linear programming problem. RSM is based on a specific form of the general linear programming problem in which all variables in the objective function and all variables in the constraints are integers. While more difficult, integer programming was required for accuracy when modeling systems with small numbers of components. For a detailed presentation of integer programming problems refer to reference 1.

As stated in the section on preparing the input, real numbers are permitted in defining the lifetime and the properties of the components. This is allowed because internally RSM converts these to integers. To do this, the lifetime is divided by the period duration and reduced to the next lowest integer. This is conservative in that the component lifetimes remain the same or are decreased. Similarly, the component properties corresponding to resources such as mass are multiplied by factors such that they retain 5 significant figures after being converted to integers.

For each component there is a set of constraints that insure that it is replaced before its lifetime expires. The additional constraints limit the number of all the components that can be installed each period. For discussion, the following is an example of a system which includes one additional constraint (besides lifetime), two calculated components and three prescribed components. The period is defined as 1 and the weighting factors are also 1. The input screens shown in Figures C1 through C3 define the problem.

Note that the components which are calculated in the optimization algorithm start with the letters 'COMP' while the prescribed components are seen as noncritical and start with the letters NCC. The constraint is designated as CONS1.

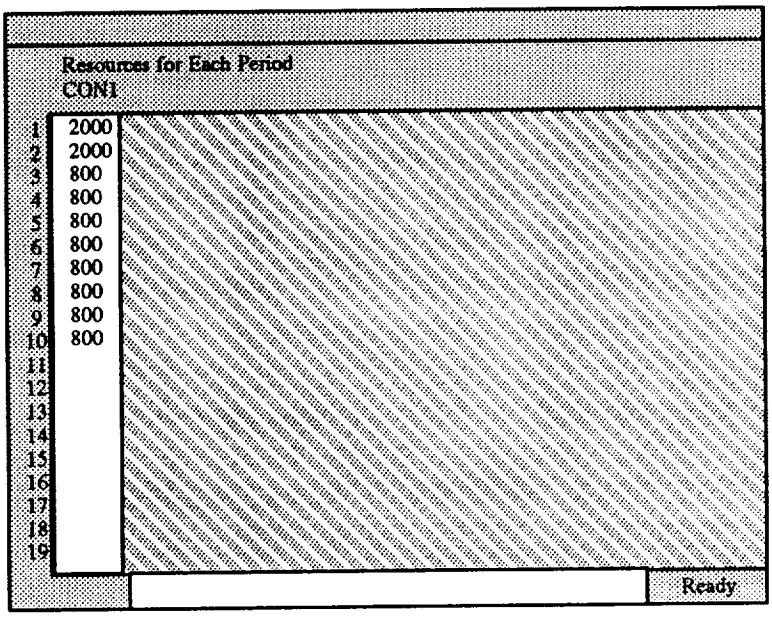


Figure C1: Resource for Each Period Input Data

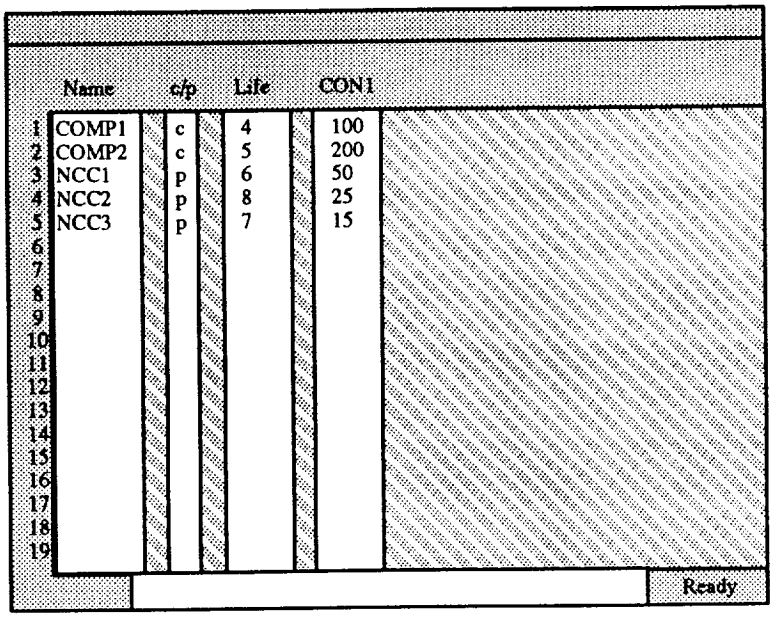


Figure C2: Component Property Input Data

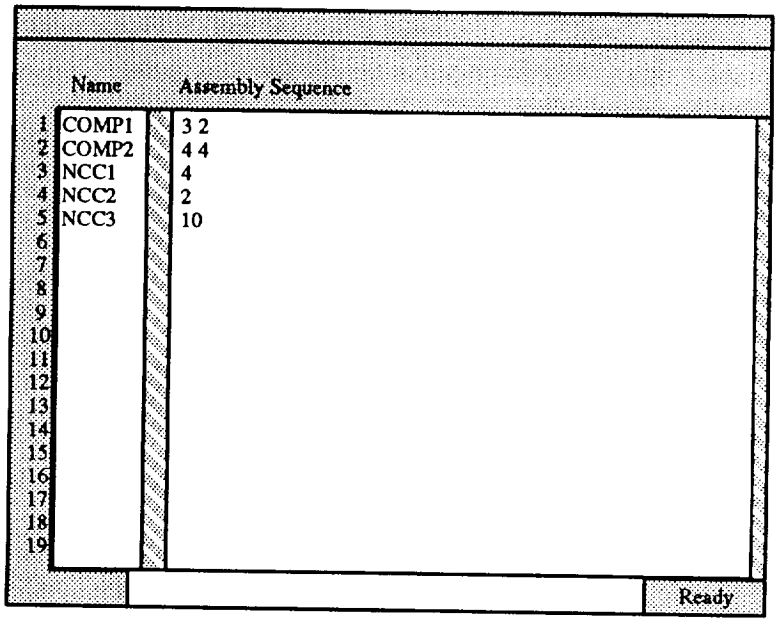


Figure C3: Component Assembly Sequence Input Data

Details of the Constraints

Let A_i denote the number of COMP1 to install/replace in period i and B_i denote the number of COMP2 to install/replace in period i.

Lifetime constraints for COMP1:

$$A_1 = 3$$
$$A_2 = 2$$
$$A_3 + A_4 + A_5 \geq 3$$
$$A_3 + A_4 + A_5 + A_6 \geq 5$$
$$A_4 + A_5 + A_6 + A_7 \geq 5$$
$$A_5 + A_6 + A_7 + A_8 \geq 5$$
$$A_6 + A_7 + A_8 + A_9 \geq 5$$
$$A_7 + A_8 + A_9 + A_{10} \geq 5$$

The first two constraints are for the assembly sequence. The three units installed in period 1 will expire by period 5, so in periods 3, 4 and 5, at least 3 units will need to be replaced; this is the third constraint. By period 6, all the units will have to be replaced; this is the fourth constraint. After this every four (the component's lifetime) periods, 5 units will have to be

replaced. The reason the constraints must overlap is to take care of units that are replaced before their lifetime has expired. Note that normally the number of periods used to replace the units is the lifetime (4 in this case), however the third constraint only can use 3 periods because of the assembly sequence.

Lifetime constraints for COMP2:

$$B_1 = 4$$
$$B_2 = 4$$
$$B_3 + B_4 + B_5 + B_6 \geq 4$$
$$B_3 + B_4 + B_5 + B_6 + B_7 \geq 8$$
$$B_4 + B_5 + B_6 + B_7 + B_8 \geq 8$$
$$B_5 + B_6 + B_7 + B_8 + B_9 \geq 8$$
$$B_6 + B_7 + B_8 + B_9 + B_{10} \geq 8$$

Once again, the first two constraints are for the assembly sequence. By period 6, the lifetime of the units installed in period 1 will have expired, so 4 units need to be replaced during periods 3, 4, 5 and 6; this is the third constraint. After this, every 5 (the lifetime) periods, all the units need to be replaced.

Additional constraint CON1:

First, the amount of the allocation used for the NCCs must be calculated. NCC1 is installed in period 1 and has a lifetime of 6, so it needs to be replaced every 6 periods (since the model is only for 10 periods, period 7 is the only replacement period. Similarly, each NCC must be accounted for.

Note that a value is not subtracted from the initial period because the model assumes that all units can be installed. Also, the user must be certain not to over–constrain the assembly sequence periods with additional constraints (i.e. to meet the assembly sequence, period 1 needs an allocation of at least 1100 and period 2 an allocation of at least 1000).

Here are the allocations after the NCCs are replaced.

Period	Allocation
1	2000
2	2000
3	800
4	800
5	800

6	800
7	800 – 4 * 50 = 600 (NCC1)
8	800 – 10 * 50 = 650 (NCC3)
9	800 – 2 * 25 = 750 (NCC2)
10	800

Now the constraint coefficients can be made. Note that the constraint coefficient for COMP1 is 100 and for COMP2 it is 200

$$100A_1 + 200B_1 \leq 2000$$
$$100A_2 + 200B_2 \leq 2000$$
$$100A_3 + 200B_3 \leq 800$$
$$100A_4 + 200B_4 \leq 800$$
$$100A_5 + 200B_5 \leq 800$$
$$100A_6 + 200B_6 \leq 800$$
$$100A_7 + 200B_7 \leq 600$$
$$100A_8 + 200B_8 \leq 650$$
$$100A_9 + 200B_9 \leq 750$$
$$100A_{10} + 200B_{10} \leq 800$$

Objective Function

The objective function is the sum of the number of each component multiplied by its objective coefficient. In the previous example, both the objective coefficients are 1 so the objective function is:

$$1A_1 + 1A_2\ 1A_3 + \ldots + 1A_9 + 1A_{10} + 1B_1 + 1B_2 + 1B_3 + \ldots + 1B_9 + 1B_{10}$$

Solving the Integer Programming Problem

Unlike linear programming where the Revised Simplex Method is the best method for all problems, integer programming does not have a method that is best for all problems; the IP solver used should depend on the type of problem that needs to be solved. In this case, all the scheduling problems produce the same general type of problem. A few different methods were tried to solve the problems, a branch–and–bound method and two all–integer cutting–plane methods. For the scheduling problems, one of the all–integer methods using a dual–simplex–like method performed significantly better than the branch–and–bound method.

Following is the set–up of the initial tableau. The first row contains the objective function of size m. The next m rows are a m by m identity matrix. The remaining rows contain the constraint matrix (without the right hand side b vector); these equations must all be greater than or equal constraints. (Note: any less than or equal constraints can be converted by multiplying both the constraint and the corresponding element in the right hand side vector by –1 and equality constraints can be converted by treating the constraint as a greater than or equal constraint and adding a less than or equal constraint). Next, insert a first column; the first m+1 entries are 0 and remaining entries are negative the right hand side vector b.

Example:

Minimize $2X1 + 6X2 + 3X3$
Such That
$X1 + 3X2 + X3 \geq 5$
$2X1 + 5X2 - 3X3 \geq 6$
$2X1 + 3X2 + 2X3 \geq 4$

Initial Tableau:

0	2	6	3
0	1	0	0
0	0	1	0
0	0	0	1
–5	1	3	1
–6	2	5	–3
–4	2	3	2

Algorithm:

For the problem:
min $\mathbf{c}^T\mathbf{x}$
such that
$\mathbf{Ax} \geq \mathbf{b}$

1. If all entries in the first column are non–negative, the problem is solved, go to step 6

2. Select the first row with a negative entry in column 1, denote it row v. Let K be the set of indices k, where $a_{vk} > 0$. If K is empty, stop – the problem has no solution.

3. Determine the index, s of the lexicographically minimum column of the columns which are a member of K.

Note: A vector is lexicographically positive if the first non–zero entry is positive. Vector **a** is lexicographically greater than vector **b** if (**a** – **b**) is lexicographically positive.

4. Find the largest μ_j that maintains $a_j - \mu_j a_s$ lexicographically positive for all $j \varepsilon K$, where a_j and a_s are the j and s columns of A. This can be done by the following method:

If a_j and a_s begin with an unequal number of zero, let $\mu_j = \infty$, other wise let e_j and e_s be the first non–zero terms in columns j and s. If e_s does not divide e_j, let $j = [e_j/e_s]$, where [a] denotes the greatest integer less than or equal to a. If e_s does divide e_j then if $a_j - (e_j/e_s)\, a_s$ is lexicographically positive then let $\mu_j = e_j/e_s$ else let $\mu_j = e_j/e_s - 1$. Let $\mu_s = 1$. Let $\lambda = \max_{(j \varepsilon K)} a_j/u_j$

5. Let $q = \{b_v/\lambda\}$ and $p_j = \{a_j/\lambda\}$, where {a} denotes the smallest integer greater than or equal to a.
Let $b = b + q\, a_s$ (where b is the first column of the tableau and a_s is the s column of the tableau)
Let $a_j = a_j - p_j\, a_s$ for $j \neq s$, where a_j is the j column of the tableau).

6. The solution is contained in the first column of the tableau. The first entry is the value of the objective function. The next m entries are the values of the variables in the objective function.

Final tableau:

10	0	2	1
2	3	–2	–1
1	–1	1	0
0	0	0	1
0	0	1	0
3	1	1	–5
3	3	–1	0

The value of the objective function is 10 and X1 = 2, X2 = 1 and X3 = 0.

Appendix D: Program Documentation

This appendix is intended to aid those who wish to expand, alter or maintain RSM code. The code is written in APL2 on an IBM PC, thus a knowledge of APL is necessary. Also a solid background in linear/integer programming is necessary. The code is readily dividable into three separate sections: the pre–processor section for entering the scheduling data, the solving routines for solving an integer programming problem, and the interface between them.

RSM is documented with extensive internal comments. This appendix describes the main functions and then presents an APL generated function name list with descriptions for most of the important functions. This is followed by a flowchart, and finally a description of the data structure.

In addition, while in the APL environment, two functions are available to aid in the understanding of the RSM program.

The first function can be used to obtain a one line explanation of a function or variable. The syntax is as follows:

EXPLAIN '*function name*'

For example:

EXPLAIN 'DATACHECK'

CHECKS DATA BEFORE SOLVING PROBLEM

A second function finds all occurrences of a string in all functions in the workspace. The syntax is as follows:

WHERE '*string*'

Input Entry Functions:

RSMΔINP is the main input control function. Two input editors are used, a table editor SCRNTAB and a multiple field input editor SCRNMFI.

Interface Between the Input Entry and the Integer Program Solver:

The interface between the two sections of code is the function FORMULATE; it takes data from the pre–processor data structures and converts it into an IP in the standard form described above. FORMULATE uses DONCC to calculate the allocations used by the prescribed components.

Integer Programming Functions

The main IP function is CUT and it uses two other functions, GETLAMBDA and GETLEXMIN. The algorithm works for minimization problems (or maximization problems with negative coefficients for the objective function – a minimization problem in disguise).

RSM APL Function Name List and Description

BARCHT
  CREATES A CHARACTER BASED BARCHART OF VALUES IN X
BARCHTΔLIM
  INSERTS LIMIT SIGN IN BAR CHART
BARCHTΔSCL
  SCALES BAR CHART Y AXIS
CCTAB
  CONVERTS COMPONENT DATA TO A TABLE FOR EDITING
CHKDATA
  RETURNS A ZERO IF DATA IS OK
CONTROL
  TOP LEVEL CONTROL FUNCTION
CUT
  SOLVE ALL-INTEGER PROBLEM
DATACHECK
  CHECKS DATA BEFORE SOLVING PROBLEM
DATAΔTYPE
  RETURNS 1 FOR EACH NUMERIC DATA MEMBER, 0 FOR EACH CHARACTER DATA
DESCRIBE
  DISPLAY PROGRAM DESCRIPTION
DIR3
  RETURNS A DOS DIRECTORY
DISPΔSOLNSCR
  DISPLAY THE SOLUTION TO THE PROBLEM
DISPΔSURPLUS
  DISPLAY THE AMOUNT OF ALLOCATIONS USED
DONCC
  CONSTRAINT ALLOCATIONS SUBTRACTING THOSE FOR PRESCRIBED COMPONENTS
EMUL8087
  INITIALIZED THE 8087 EMULATOR
FILEΔDELT
  DELETES A SYSTEM FILE
FILEΔDIR
  RETURNS A FORMATTED FILE LISTING OF QUALIFIED NAMES USING LONG NAMES
FILEΔINFO
  DOS DIR. USING AUX PROCESSOR 103 (ORD=0 FOR CHRON,ORD=1 ALPHABET ORDER)
FILEΔLOAD
  RETRIEVES SYSTEM INPUT FROM A FILE
FILEΔRENM
  RENAMES A SYSTEM INPUT FILE
FILEΔSAVE
  SAVES LISTED VARIABLES TO A QUALIFIED FILE AND UPDATES DIRECTORY
FORMLTΔ1
  CALLED BY FORMULATE, NEEDS EXTERNAL PASS OF: RHSIDE TEMP

FORMLTΔR
GENERATES RESOURCE CONSTRAINT MATRICES FOR FORMULATE
FORMULATE
MAKES LIFETIME CONSTRAINTS FOR A COMPONENT
FORMΔSTAT
DISPLAY STATUS OF THE FORMULATION
FΔERASE
AP103 FUNCTION TO ERASE A FILE
GETLAMBDA
RETURNS LAMBDA VALUE FOR CUTTING METHOD
GETLEXMIN
RETURNS THE LEXICOGRAPHICALLY LEAST COLUMN IN THE MATRIX B
IN
EMULATES THE)IN COMMAND
INITΔNUM
CALCULATES NUMBER OF COMPONENTS, CONSTRAINTS, AND PERIODS
INPΔCOMP
ENTER COMPONENT INFORMATION
INPΔCOMPΔ1
ENTER COMPONENT INFORMATION
INPΔCOMPΔ2
ENTER COMPONENT ASSEMBLY SEQUENCE
INPΔCOMPΔ3
ENTER COMPONENT WEIGHTING FACTOR
INPΔCOMPΔCHK
CHECKS COMPONENT DATA TABLE AND RETURNS ERROR MESSAGE
INPΔDELT
DELETES AN INPUT FILE
INPΔFILE
INPUT DATA FILE MANAGER
INPΔLOAD
LOADS AN INPUT DATA FILE
INPΔRC
ENTER RESOURCES AND COMPONENTS
INPΔRENM
RENAMES AN INPUT FILE
INPΔRSR
ENTER RESOURCE INFORMATION
INPΔRSRΔ2
UPDATE COMPONENT DATA
INPΔRSRΔCHK1
CHECKS RESOURCE NAMES AND PERIOD DURATION AND RETURNS ERROR MESSAGE
INPΔSAVE
SAVE A SYSTEM INPUT FILE
MATHCOPR
TURNS ON MATH COPROCESSOR EMULATION IF NEEDED

OUT
  EMULATES THE)OUT COMMAND
RESΔDELT
  DELETES A RESULTS FILE
RESΔDISP
  DISPLAY THE RESULTS
RESΔFILE
  RESULTS FILE MANAGER
RESΔLOAD
  LOADS A RESULT FILE
RESΔPRT
  PRINT THE RESULTS
RESΔRENM
  RENAMES A RESULTS FILE
RESΔSAVE
  SAVE A RESULTS FILE
RSM
  MAIN FUNCTION FOR RESUPPLY SCHEDULING MODEL
RSMΔEND
  PREPARE TO LEAVE PROGRAM
RSMΔERR
  OVERALL ERROR HANDLING FOR RSM
RSMΔEXTRTNS
  DUMMY FUNCTION FOR FLOW CHART DOCUMENTATION
RSMΔINIT
  INITIALIZES VARIABLES FOR RSM
RSMΔINP
  ENTER AND FILE INPUT
RSMΔRES
  DISPLAY, PRINT AND FILE RESULTS
RSMΔSOLVE
  SOLVE THE PROBLEM
SCRNMFI
  MULTIPLE FIELD INPUT SCREEN
SCRNTAB
  SCREEN TABLE OUTPUT FIELD WITH HEADER AND SCROLLING (DKS VERSION)
SOLVΔSTAT
  DISPLAY STATUS OF THE SOLUTION

Flowchart of RSM APL Workspace

RSM
:MATHCOPR
:EMUL8087
:RSMΔERR
:CONTROL
:RSMΔINIT
:RSMΔEXTRTNS
:DESCRIBE
:RSMΔINP
:INPΔRC
:INPΔRSR
:SCRNMFI
:DATAΔTYPE
:INPΔRSRΔCHK1
:INPΔRSRΔ2
:SCRNTAB
:INPΔCOMP
:INPΔCOMPΔ1
:CCTAB
:SCRNTAB
:INPΔCOMPΔCHK
:INPΔCOMPΔ2
:SCRNTAB
:DTZ
:INPΔCOMPΔ3
:SCRNTAB
:INPΔFILE
:INPΔLOAD
:FILEΔLOAD
:INPΔSAVE
:FILEΔSAVE
:DIR3
:FILEΔINFO
:IN
:FILEΔDIR
:OUT
:INPΔDELT
:FILEΔDELT

```
            :FΔERASE
            :IN
            :OUT
        :INPΔRENM
            :FILEΔRENM
            :IN
            :OUT
:RSMΔSOLVE
    :DATACHECK
        :DTZ
    :INITΔNUM
    :CHKDATA
    :FORMULATE
        :FORMΔSTAT
        :FORMLTΔ1
        :FORMΔSTAT
        :DONCC
        :FORMLTΔR
    :CUT
        :GETLAMBDA
            :GETLEXMIN
        :SOLVΔSTAT
:RSMΔRES
    :RESΔDISP
        :DISPΔSURPLUS
            :BARCHT
                :BARCHTΔSCL
                :BARCHTΔLIM
        :DISPΔSOLNSCR
            :BARCHT
                :BARCHTΔSCL
                :BARCHTΔLIM
    :RESΔPRT
    :RESΔFILE
        :RESΔLOAD
            :FILEΔLOAD
            :INITΔNUM
        :RESΔSAVE
            :FILEΔSAVE
```

```
            :DIR3
                :FILEΔINFO
            :IN
            :FILEΔDIR
            :OUT
        :RESΔDELT
            :FILEΔDELT
                :FΔERASE
                :IN
                :OUT
        :RESΔRENM
            :FILEΔRENM
                :IN
                :OUT
:RSMΔEND
```

Data Structure Formats

The two main data structures are CONSLIST and COMPSTRUC. The following is the set–up for the ith entry of each data structure.

Each entry of CONSLIST contains three elements.

⊃(⊃CONSLIST[I])[1] – character vector containing the constraint's name
⊃(⊃CONSLIST[I])[2] – numeric vector containing maximum allocation for each period
⊃(⊃CONSLIST[I])[3] – numeric vector containing maximum allocation after the prescribed values are calculated

Each entry of COMPSTRUC contains six elements

⊃(⊃COMPSTRUC[I])[1] – character vector containing component's name
⊃(⊃COMPSTRUC[I])[2] – numeric scalar containing the component's weighting factor
⊃(⊃COMPSTRUC[I])[3] – numeric scalar containing component's lifetime
⊃(⊃COMPSTRUC[I])[4] – character scalar designating calculated or prescribed
⊃(⊃COMPSTRUC[I])[5] – numeric vector containing component's assembly sequence
⊃(⊃COMPSTRUC[I])[6] – numeric vector containing component's constraint coefficients

References

1. Greenberg, Harold, *Integer Programming,* Academic Press, Inc., New York, NY, 1971

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM -104380		

4. Title and Subtitle
RSM 1.0 User's Guide
A Resupply Scheduler Using Integer Optimization

5. Report Date
May 1991

6. Performing Organization Code

7. Author(s)
Larry A. Viterna, Robert D. Green, and David M. Reed

8. Performing Organization Report No.
E -6185

10. Work Unit No.
474-12-10

9. Performing Organization Name and Address
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135 - 3191

11. Contract or Grant No.

13. Type of Report and Period Covered
Technical Memorandum

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, D.C. 20546 - 0001

14. Sponsoring Agency Code

15. Supplementary Notes
Larry A. Viterna and Robert D. Green, NASA Lewis Research Center; David M. Reed, Wittenberg University, Springfield, Ohio 45501 and Summer Student Intern at NASA Lewis Research Center. Responsible person, Larry A. Viterna, (216) 433-5398.

16. Abstract
RSM (Resupply Scheduling Model) is a PC based, fully menu-driven computer program. It uses integer programming techniques to determine an optimum schedule to replace components on or before a fixed replacement period, subject to user defined constraints such as transporation mass and volume limits or available repair crew time. Principal input for RSM includes component properties such as mass and volume and an assembly sequence. Resource constraints are entered for each period corresponding to the component properties. Though written to analyze the electrical power system on the Space Station Freedom, RSM is quite general and can be used to model the resupply of almost any system subject to user defined resource constraints. This report presents a step by step procedure to preparing the input, performing the analysis and interpreting the results. Instructions for installing the program and information on the algorithms are contained in the appendices.

17. Key Words (Suggested by Author(s))
Scheduling
Resource allocation
Optimization

18. Distribution Statement
Unclassified - Unlimited
Subject Category 16

19. Security Classif. (of the report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	44	A03

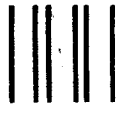
*For sale by the National Technical Information Service, Springfield, Virginia 22161

National Aeronautics and
Space Administration

Lewis Research Center
Cleveland, Ohio 44135

FOURTH CLASS MAIL

ADDRESS CORRECTION REQUESTED



Official Business
Penalty for Private Use $300

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA 451

NASA